

Robotics Research Technical Report

On Curve Matching

by

Haim Wolfson

Technical Report No. 256

Robotics Report No. 86

November, 1986

c.2
NYU COMPSCI TR-256
Wolfson, Haim
On curve matching.

New York University
Courant Institute of Mathematical Sciences

Computer Science Division
251 Mercer Street New York, N.Y. 10012



On Curve Matching

by

Haim Wolfson

Technical Report No. 256
Robotics Report No. 86
November, 1986

New York University
Dept. of Computer Science
Courant Institute of Mathematical Sciences
251 Mercer Street
New York, New York 10012

Work on this paper has been supported by Office of Naval Research Grant N00014-82-K-0381, National Science Foundation CER Grant DCR-83-20085, and by grants from the Digital Equipment Corporation and the IBM Corporation.

On Curve Matching

Haim Wolfson

New York University
Robotics Research Laboratory
Dept. of Computer Science
Courant Institute of Mathematical Sciences
251 Mercer Street
New York, N.Y. 10012

ABSTRACT

Three algorithms to find the longest common subcurve of two 2-D curves are presented. Their common denominator is the conversion of the curves into "curvature strings" and application of string matching techniques. The algorithms vary in their theoretical complexity and robustness. The most efficient is of complexity $O(n)$, where n is the number of the sample points on both curves. The algorithms can be easily extended to the 3-D case. Experimental results in 2-D are included.

1. Introduction.

The problem of finding the the best fit between two curves is of central importance in computer vision. A solution of this problem is required in various assembly problems, where a robot is required to put two pieces together along their matching boundary, e.g. "puzzle assembly" (see [F-G],[R-B], [B-W],[K-L-S-W]). Another possible application for such an algorithm is in recognition of partially occluded objects in an overlapping scene (see [T-M-V],[K-S-S-S]). Since two dimensional objects are completely described by their closed boundary curves, the detection of partially occluded objects participating in a composite scene can be done by

Work on this paper has been supported by Office of Naval Research Grant N00014-82-K-0381, National Science Foundation Grant No. NSF-DCR-83-20085, and by grants from the Digital Equipment Corporation, and the IBM Corporation.

matching the boundary curve of the scene with the boundary curves of "candidate" objects, and trying to find out whether they have a "long enough" matching subcurve. For this specific problem, however, computationally more efficient techniques have been developed (see [H-W]).

A number of algorithms are available to solve the curve matching problem under the restrictive assumption that one curve to be matched is a proper subcurve of the other. One such algorithm due to Schwartz and Sharir (see [S-S]) has proved to be very accurate, as indicated by its successful application in a technique for visual assembly of a 200-piece puzzle (see [K-L-S-W]). The complexity of this algorithm is $O(n \log n)$, where n is the number of sample points on the longer curve.

The extended algorithms presented in this note are intended to solve the more general curve matching problem:

given two curves, find the longest matching subcurve which appears in both curves.

We assume, of course, that such a subcurve does exist.

One of the techniques to be described is of complexity $O(n)$, where n is the number of sample points on the longer curve, so it improves the complexity of the Schwartz-Sharir algorithm for the proper subcurve case, as well.

Our approach can be summarized as follows :

Step A : Represent both curves by *characteristic strings* which represent local translationally and rotationally invariant features; specifically, we shall represent every curve by a sequence of its approximate local curvatures at given sample points.

Step B : Find the longest common substring of the two characteristic strings.

Step C : Take the substring produced by *Step B* , go back to the original curves and match the two subcurves which correspond to this substring using the existing Schwartz-Sharir subcurve matching algorithm.

We shall present three techniques, based on this idea, together with an analysis of their complexity and robustness. Although a certain trade-off between

robustness and the theoretical complexity of the various techniques will be noted, nevertheless we believe that the third algorithm presented below is the most advantageous, not only because of its robustness, but also because its complexity for curves of "practical" length is quite acceptable.

The earlier studies of curve matching mentioned above employed various heuristics to divide curves into subcurves as preparation for the application of proper subcurve matching algorithms. These heuristics find points at which curves turn through sharp angles (called "breakpoints" in [K-S-S-S], "critical points" in [R-B]). The curvature strings used in our new algorithm take these critical points into account automatically, and thus extend these earlier heuristic approaches to cases in which changes in curvature are not so sharp.

2. Definition of the problem.

We consider two planar curves, having at least one "sufficiently long" common subcurve. The data describing the curves is assumed to be noisy.(In our experiments curves are extracted as boundary curves of 2-D objects using a digitizing camera.) We want to find the longest common subcurve of these two curves. Once having this subcurve, we want to find the angle and displacement by which we must rotate and translate one of the curves to make it fit the other curve along their common portion.

3. Preprocessing.

We begin with three major preprocessing steps :

- 1) Planar pieces are photographed by a black and white RCA 2000 camera, and the pictures are digitized and thresholded to get a binary image for each piece.
- 2) The boundary of each piece is extracted from the binary image. These boundary curves are our "experimental" curves.
- 3) A *smoothing* procedure is applied to each curve. We use the procedure which is described in [S-S]. Basically, this expands the noisy curve to a narrow strip

defined by a certain threshold value ϵ and then finds the shortest path lying in this 2ϵ -wide strip. This gives a polygonal approximation of each observed curve.

4. Conversion of curves into characteristic strings

Since all the algorithms that we are going to present are based on conversion of curves into numerical sequences (the first two algorithms will require strings of integers, taken from a finite set, while the third one require strings of real numbers), we first discuss this conversion problem.

Our aim is to represent any curve C by a string of reals or integers $(c_i)_{i=1}^n$. Since these strings will be compared to achieve subcurve matching, we want the numbers $(c_i)_{i=1}^n$ to encode characteristics of the curve which are :

- i) local,
- ii) translationally and rotationally invariant,
- iii) stable in the sense, that small changes in the curve will have small effects (or no effect at all) on the sequence $(c_i)_{i=1}^n$,

a further desirable, but less essential property is :

- iv) an approximation to an observed curve can be reconstructed from its characteristic string.

One "natural" feature which satisfies these conditions is the pointwise *curvature* of a curve (see Chapter II of [S]). It is well known that there is a one to one correspondence between a regular curve (modulo translation and rotation) and its curvature function (which is a continuous function of its arclength). For our applications, however, we must deal with noisy polygonal representations of curves, making it impossible to compute curvatures either accurately, or at every point of the curve. Thus we must work with an approximation of the curvature, calculated at discrete points of the curve, to get a data sequence $(c_i)_{i=1}^n$ having the desired properties.

Let $\kappa(s)$ be the curvature function of a curve C , where s denotes arclength along the curve. $\kappa(s)$ is the derivative of the tangent angle $\theta(s)$ to the curve, which we parametrize as a function of its arclength. Hence, in order to approximate the curvature, we first build the so called *arclength versus turning angle graph* of the curve C . (Since after our smoothing procedure we have a polygonal approximation of the observed curve, this is a step function.) Then we sample this graph at equally spaced points, and at every such point s_i ($i = 1, \dots, n$) we compute the difference

$$\Delta\theta(s_i) = \theta(s_i + \Delta s) - \theta(s_i)$$

(To make the method more robust we actually compute an averaged difference

$$\phi_i = Av\Delta\theta(s_i) = \frac{1}{k} \sum_{j=0}^{k-1} \Delta\theta(s_i + j\delta)$$

Detailed choice of the parameters Δs , k , δ is based on experimental considerations.)

Remark: Note that the averaged differences (ϕ_i) satisfy the conditions (i)-(iii) which we required from local curve characteristics. Even though they were chosen only as an approximation of the curvature, they posses by themselves the most essential properties for such a characteristic.

It is obvious that if the curve does not have double points (this is the case with parts of boundaries of connected simple regions) then the elements of the sequence $(\phi_i)_{i=1}^n$ are real numbers in the interval $[-2\pi, 2\pi]$.

If we want to get a string of integers instead of real numbers, we can simply multiply the values $(\phi_i)_{i=1}^n$ by a magnification constant M and take the closest integer, giving the string

$$\Phi_i = \text{closest integer} (M\phi_i)$$

This gives a string of integers belonging to a finite alphabet of size at most $[4\pi M]$.

Plainly the numbers Δs and M used have a big impact on the accuracy with which the quantities $(\Phi_i)_{i=1}^n$ approximate the true curvature of an initially smooth model curve. However our aim is not to reconstruct the curve from the integer string

$(\Phi_i)_{i=1}^n$, but to match these strings for two different curves.

5. The general scheme of the algorithms

All three algorithms to be described in this section share the following three basic steps :

Step A : Translate both curves into curvature strings.

The complexity of *Step A* is $O(n)$, where n is the number of sample points on both curves.

Step B : Find the longest common substring, or several long common substrings of the two curvature strings.

The complexity of this step is analyzed in the following sections.

At first glance this step would seem not to be robust, since it depends on the accuracy of *Step A* and on several choices of detail besides. However, we should remember that the aim of this step in all three cases is quite limited: all we want is to locate approximate starting points and endpoints of several long candidate subcurves for matching. Matching itself will use the robust subcurve matching algorithm, which constitutes:

Step C : Extract a subcurve from one of the curves using the starting and end points obtained in *Step B* , and match this subcurve to the other curve using the Schwartz-Sharir subcurve matching algorithm (see [S-S], also Appendix I). This finds the rotation and translation of the best L_2 fit of a subcurve to a specified model curve.

The Schwartz-Sharir algorithm is of complexity $O(n \log n)$, where n is the number of sample points on the longer curve (notice, that in our case this number is of the same magnitude as the length of the curvature strings). However, the factor of $\log n$ appears in this algorithm because no knowledge about the displacement of the proper subcurve on the longer curve is assumed (see Appendix I). Our case is different: since we know the starting points and endpoints of the match between the curvature strings, the appropriate displacement can be calculated more efficiently.

(But to make the fit more accurate, we also check a small number of displacements to both sides of the computed location.) Thus we can find the desired rotation and translation in $O(n)$ operations.

6. Algorithms based on integer string matching procedures

In this section we present two algorithms which use efficient string matching techniques. These algorithms require strings of characters drawn from a finite alphabet. For this reason, initially given curves must be transformed into integer curvature strings in the variant of *Step A* used with these algorithms. The elements of these strings belong to an alphabet of size at most $\lceil 4\pi M \rceil$ (see section 4).

Suppose that we are given two curves C and $\tilde{C}$ and have converted them into integer strings denoted by $(\Phi_i)_{i=1}^n$ and $(\Psi_i)_{i=1}^m$ respectively. Our two proposed algorithms are then as follows.

Algorithm I

Step B : Find the longest matching substring between these two strings using Weiner's algorithm (see [W],[C-S]). It's complexity for a finite alphabet is $O(n)$. Let $\Phi_1 = (\Phi_i)_{i=s}^t$ and $\Psi_1 = (\Psi_i)_{i=u}^v$ be the matching substrings obtained. Then apply

Step C : Compute the starting point and endpoint of Φ_1 on the original curve C , thereby finding the subcurve C_1 which has Φ_1 as its characteristic substring. Use the Schwartz-Sharir subcurve matching algorithm to find the rotation and translation which gives the best L_2 fit of C_1 into $\tilde{C}$. As explained in the previous section this step is of complexity $O(n)$. Check to see if the L_2 distance of this fit is under a desired threshold, and, if so, apply the computed rotation and translation to match the curves.

This algorithm is quite efficient, but may miss the longest matching subcurve, because of roundoff errors, since conversion of a real number string into an integer string by taking the closest integer can cause two otherwise close numbers to be

converted into different integers. This can cause long matching subportions of two curves to be converted into characteristic strings which match only with interruptions, and, hence can obtain an inappropriate longest matching substring. However, such mismatches can be made rare, since we can make the probability of such an event reasonably small. Specifically, if the precision of the calculation of the real number curvature string $(\phi_i)_{i=1}^n$ is ϵ , and if the numbers (ϕ_i) are uniformly distributed in the interval $[-2\pi M, 2\pi M]$, then the probability that two ϵ -close numbers will be converted to different integers is $2\epsilon M$.

However, in order to make our algorithm more robust, we can look not for the longest matching substring between the two characteristic strings, but for the K longest matching substrings for some constant K , which is given in advance, or for all matching substrings of length at least l , for some prescribed l . This suggests

Algorithm II

Step A as in *Algorithm I*.

Step B : For a given K use the algorithm of Karp, Miller and Rosenberg (see [K-M-R]) to find the K longest matching substrings between the two characteristic strings. The complexity of this step is $O(n \log n)$.

Given a collection of "reasonably long" matching substrings, we can then check consistency between their shifts and discover if they are parts of the same "longest matching substring" with local mismatches. In this way we can also discover errors caused by a small variance of parametrization, which causes an insertion or deletion of a character in the characteristic string.

After completion of these "local" modifications we pass the set of the long matching substrings to the last step of the algorithm, which must be modified as follows.

Step C : For every substring obtained from *Step B* find the appropriate translation and rotation using the Schwartz-Sharir algorithm (see description of *Algorithm I*), rotate and translate the curves accordingly, and check the longest matching

subcurve between the two curves, given the rotation and translation. This subcurve is found by simply checking the (x,y) coordinates of corresponding points on the curves and demanding that the distance between the points should be less than a certain threshold value ϵ . This final check works with points on the curves themselves, and not the less accurate curvature values at these points; hence is quite robust. The result giving the longest matching subcurve (allowing minor mismatches) is chosen as the final solution. The complexity of this step is $O(Kn)$, where K is the number of results passed from *Step B*, which can be fixed in advance. Thus the complexity of *Algorithm II* is $O(n \log n)$, if K is held to a value less than $\log n$.

7. An algorithm based on matching of real sequences

In order to improve the robustness of the algorithms, one wants a method which does not require truncation of the curvature strings and exact coincidence of the truncated numbers at a given point. Instead we would like to be able to say that two elements of a curvature string are equal, if there is at most an ϵ -difference between them, where ϵ represents a measure of accuracy. To achieve this goal *Step B* requires a string matching algorithm, which allows non-equality comparison of string elements. The algorithm that we describe in this section is of $O(n^2)$ worst case complexity, however it is quite efficient in practical applications.

Algorithm III

Step A : Transform both curves into real number curvature strings as described in Section 4.

Step B : This step differs from those used in the previous algorithms, so we describe it in more detail. Our first goal is to find shifts of one string relative to the other which produce lots of ϵ -coincidences between the string elements. Accordingly, we define an accumulator for each of the possible relative shifts (their number is of order $2n$), zero these accumulators, and apply the following procedure :

B1). Sort both curvature strings together, by the magnitude of the different elements; sorting must return the string number and the index in the original sequences of each element entering into the sorting process.

B2). Using the sorted sequence, find, for each element c_i of the first original sequence, all those elements of the second sequence $\tilde{c}_j$ which are in its ϵ -neighbourhood. For each such pair compute the relative shift between the sequences $d=i-j$ and advance the accumulator for d by one.

This step may be visualized as sliding an ϵ -wide window along the sorted sequence and computing the relative shift only for those elements which happen to be in the window and belong to different initial sequences.

This procedure generates a list of shifts which accumulate the most votes. Although the procedure may require n^2 operations in the worst case, the average number of operations will often be of the order ϵn^2 (assuming uniform distribution of the string elements in the interval $[-2\pi, 2\pi]$).

Since we are not dealing with actual curves, but with polygonal approximations, every shift scoring high in this procedure will have a set of neighbouring "strong" shifts. Hence, one actually wants to compute intervals of "strong" shifts and to choose one or more representative shifts within each such interval, e.g. the middle and the peak shift in the interval. Since *Step C* tries a number of displacements (see *Algorithm II*) we are not concerned with getting the "exact" shift.

A prescribed number K of "best" shifts is then passed to the next stage of our algorithm. K can be given in advance or decided dynamically and based on the gaps between the numbers of votes, scored by different shifts.

B3). For each shift, obtained from the previous stage, we compare the curvature strings and look for the longest substring within which differences of at most ϵ appear between the individual elements. This procedure can also allow for sporadic mismatches. The starting point and endpoint of this longest matching substring are passed to *Step C*.

The worst case complexity of *Step B* is $O(n^2)$, but its average case complexity is $O(n \log n) + O(\epsilon n^2)$.

Step C is performed in the same way as described for *Algorithm II*. For every result which passed *Step B* we find the best rotation and translation for a small number of neighbouring candidate shifts, and check on the original curves, finding the common matching portion of the curves, given this rotation and translation. This final check is done in the plane and uses the Euclidean distance between matching points. The result which gives the longest matching subcurve is taken to be the best. The complexity of this final step is again $O(n)$.

Algorithm III has the advantage of not introducing unnecessary approximations. It is also designed in a flexible way, so that a collection of candidate solutions is passed from step to step, thus reducing the probability that the correct solution will be eliminated at an early stage of the algorithm. The final decision between the "candidate" solutions is done only in the last stage, where all the "candidate" subcurves are actually tested in the plane by a robust subcurve matching algorithm.

8. Experimental results

We have implemented both *Algorithm I* and *Algorithm III*. In practice *Algorithm III* is not only robust, but also fairly efficient.

As was mentioned in section 3, all the curves were obtained experimentally rather than by simulation.

Fig. 1-12 show examples of curves which were matched by these algorithms, giving the two curves matched and their superposition along the matching part. The results of Fig. 1-3 were obtained by both algorithms. All other results were obtained by *Algorithm III*. No serious attempt was made to improve the performance of the programs by optimizing their parameters. Our objective was to validate the main concepts of the proposed approach.

The curves used as examples were extracted as boundaries of 2-D pieces and groups of pieces forming a composite overlapping scene. As can be seen the pieces

of Fig.1 belong to the same scene. So do the pieces of Fig.2-6 and Fig.7-12. In all these examples one of the curves represents the boundary of the composite scene and the other curves are boundaries of individual pieces participating in the scene. Thus these examples also show, how our algorithm can be used for object recognition in overlapping scenes.

9. Future plans

The algorithm described here can be applied to the 3-D curve matching as well. As well known from Differential Geometry (see [S]), the inherent characteristics of a 3-D curve are curvature and torsion, so that the natural extension of our method would be to use a characteristic string which combines both features. However, since we do not need to reconstruct curves but only to compare them locally, it may be enough for all practical purposes to use characteristic strings based on curvature only, curvature being a more stable geometric property. The robustness of the method, will, of course, depend on the quality of 3-D curve information available.

10. Conclusion

We have described several methods for finding the longest matching subcurve between two curves. One of these methods has complexity $O(n)$, where n is the number of sample points on the curves. It is based on conversion of the curves into "curvature strings" and using efficient "substring matching" algorithms. We have also sketched another version of the algorithm based on similar ideas, which may be more robust. This algorithm's complexity is however $O(n \log n)$. The third version of our algorithm is significantly more robust than the other two algorithms presented and was successfully tested on various examples. Its worst case complexity is $O(n^2)$, but its average performance is $O(n \log n) + O(\epsilon n^2)$, where ϵ is the measure of our accuracy.

The methods described have a natural extension to the 3-D case.

Acknowledgments

The author is grateful to Jacob T.Schwartz and Micha Sharir for helpful discussions.

APPENDIX I

Proper subcurve matching algorithm

We give a short overview of the 2-D curve matching algorithm of [S-S]. For a more detailed exposition see [S-S].

Take two curves C and C' in the plane and assume that C is a translated and rotated subcurve of C' . Both C and C' are assumed to have been smoothed (i.e. they are polygonal approximations of the original curves) and parametrized by arc length s . The matching we seek calls for determination of the offset s_0 and the Euclidean transformation E for which the curves $EC(s)$ and $C'(s+s_0)$ are closest to one another in the L^2 norm. To be more specific, we represent each of the curves C , C' by a sequence of evenly spaced points on it, and let these sequences be $(\mathbf{u}_j)_{j=1}^n$ and $(\mathbf{v}_j)_{j=1}^m$ respectively. Assume first that both curves have the same starting point (i.e. $s_0=0$ and, hence, $m \geq n$). Matching thus amounts to finding a Euclidean motion E of the plane which will minimize the l^2 distance between the sequences $(E\mathbf{u}_j)_{j=1}^n$ and $(\mathbf{v}_j)_{j=1}^n$:

$$\Delta = \min_E \sum_{j=1}^n |E\mathbf{u}_j - \mathbf{v}_j|^2$$

To simplify the calculation, first translate C so that

$$\sum_{j=1}^n \mathbf{u}_j = 0$$

Next write E as $E\mathbf{u} = R_\theta \mathbf{u} + \mathbf{a}$, R_θ denoting a counterclockwise rotation by θ .

In such case, as it is shown in [S-S], the best match is obtained when

$$\mathbf{a} = \frac{1}{n} \sum_{j=1}^n \mathbf{v}_j$$

and θ is the negation of the polar angle of $\sum u_j \bar{v}_j$, where the vectors $\mathbf{u}_j$, $\mathbf{v}_j$ are regarded as complex numbers u_j , v_j . The least-square distance for this best match is given by

$$\Delta = \sum_{j=1}^n |\mathbf{v}_j|^2 - \frac{1}{n} \left| \sum_{j=1}^n \mathbf{v}_j \right|^2 + \sum_{j=1}^n |\mathbf{u}_j|^2 - 2 \left| \sum_{j=1}^n u_j \bar{v}_j \right| \quad (*)$$

If the curves do not have the same starting point, we have to match the sequence $(\mathbf{u}_j)_{j=1}^n$ to each of the contiguous subsequences $(\mathbf{v}_{j+d})_{j=1}^n$ of the sequence $(\mathbf{v}_j)_{j=1}^m$, for $d = 0, \dots, m-n$.

For each such d (*) thus becomes

$$\Delta(d) = \sum_{j=d+1}^{d+n} |\mathbf{v}_j|^2 - \frac{1}{n} \left| \sum_{j=d+1}^{d+n} \mathbf{v}_j \right|^2 + \sum_{j=1}^n |\mathbf{u}_j|^2 - 2 \left| \sum_{j=1}^n u_j \bar{\mathbf{v}}_{j+d} \right|$$

We seek the minimum of the values $\Delta(d)$, $d = 0, \dots, m-n$, which can be found in time $O(m \log m)$, using the fast Fourier transform algorithm for computing the convolutions $\sum_{j=1}^n u_j \bar{\mathbf{v}}_{j+d}$.

REFERENCES

- [B-W] Burdea G. and Wolfson H., *Automated Assembly of a Jigsaw Puzzle Using the IBM 7565 Robot*, Tech. Rep. No.188, Nov. 1985, Comp. Sc. Div., Courant Inst. of Math., NYU.
- [C-S] Chen M.T. and Seiferas J., *Efficient and Elegant Subword-Tree Construction*, Univ. of Rochester Comp. Sc. Research Review 1983-1984, 10-14.
- [F-G] Freeman H. and Garder L., *Apictorial Jigsaw Puzzles : The Computer Solution of a Problem in Pattern Recognition*, IEEE Trans. on Electronic Comp., Vol EC-13, April 1964, No.2, 118-127.
- [H-W] Hong J.W. and Wolfson H., *On Footprints*, in preparation.
- [K-S-S-S] Kalvin A., Schonberg E., Schwartz J.T., and Sharir M., *Two Dimensional Model Based Boundary Matching Using Footprints*, Tech. Rep. No.162, May 1985, Comp. Sc. Div., Courant Inst. of Math., NYU.
- [K-L-S-W] Kalvin A., Lamdan Y.Schonberg E., and Wolfson H., *Solving Jigsaw Puzzles Using Computer Vision*, Tech. Rep. No.205, Feb. 1986, Comp. Sc. Div., Courant Inst. of Math., NYU.
- [K-M-R] Karp R.M., Miller R.E., and Rosenberg A.L., *Rapid Identification of Repeated Patterns in Strings, Trees and Arrays*, Proc. 4'th Annual ACM Sym. on Theory of Computing,(1972), 125-136.
- [R-B] Radack G.M. and Badler N.I., *Jigsaw Puzzle Matching Using a Boundary-Centered Polar Encoding*, Computer Graphics and Image Processing 19,(1982), 1-17.
- [S] Stoker J.J., *Differential Geometry*, Wiley-Interscience, 1969.
- [S-S] Schwartz J.T., and Sharir M., *Identification of Partially Obscured Objects in Two Dimensions by Matching of Noisy "Characteristic Curves"*, Tech. Rep. No.165, June 1985, Comp. Sc. Div., Courant Inst. of Math., NYU (to appear in the Int. J. of Robotics Research).

- [T-M-V] Turney J.L., Mudge T.N., and Volz R.A., *Recognizing Partially Occluded Parts*, IEEE Trans. on Pattern Analysis and Machine Intelligence, Vol. 7, No. 4, (July 1985), 410-421.
- [W] Weiner P., *Linear Pattern Matching Algorithms*, 14'th Annual Sym. on Switching & Automata Th.(Iowa), IEEE Comp. Soc., (1973), 1-11.

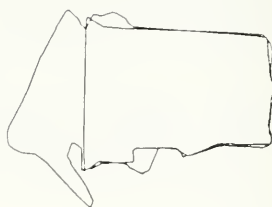


Figure 1

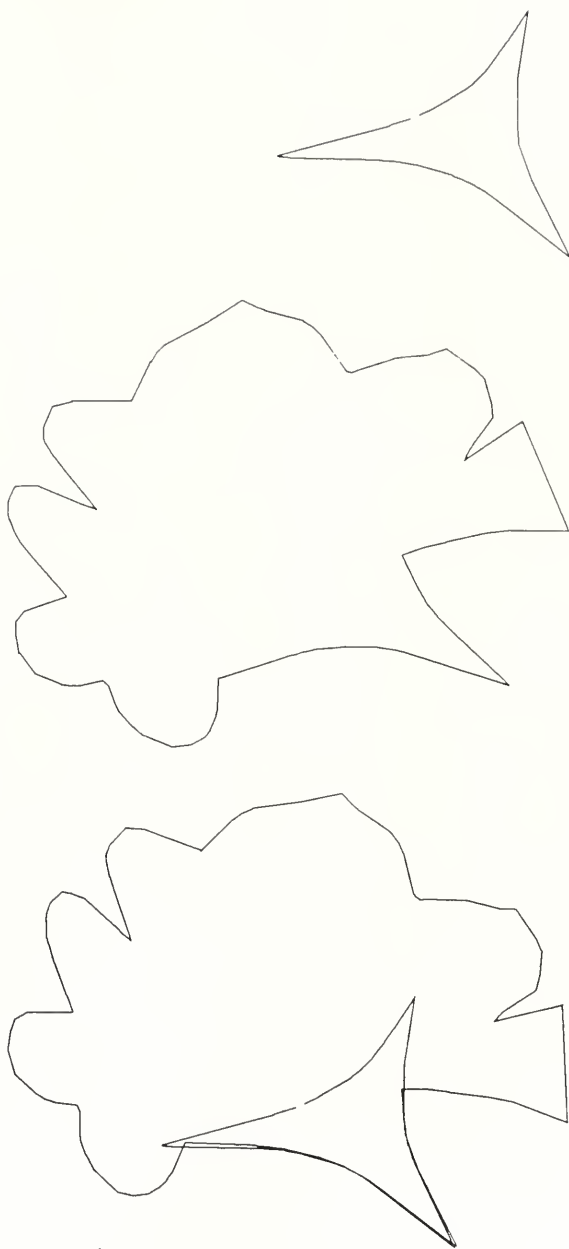


Figure 2

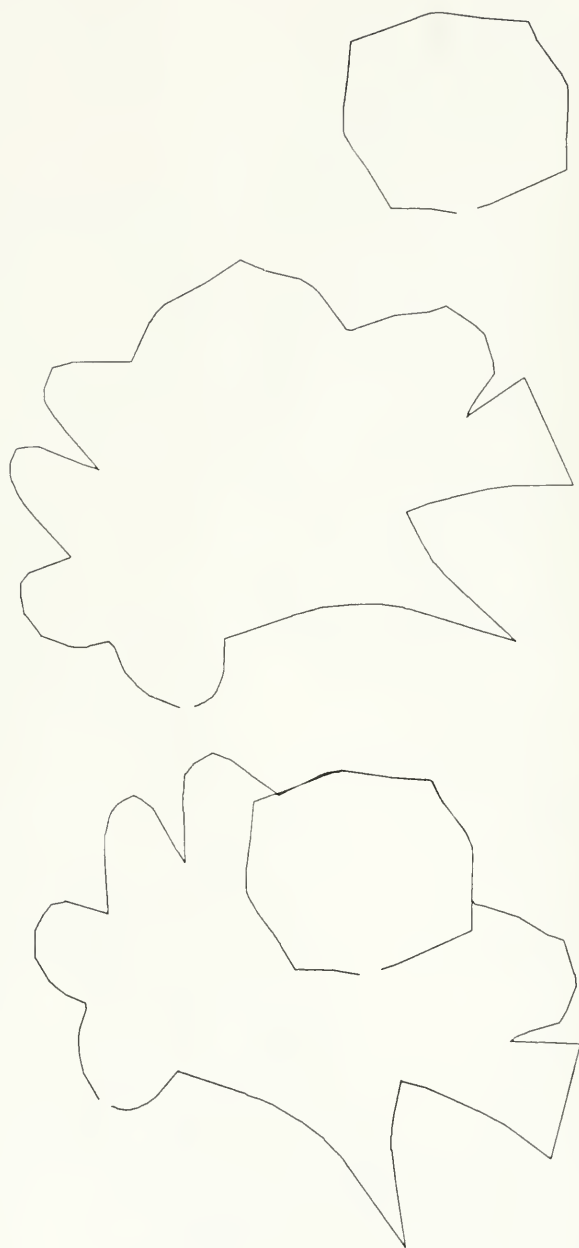


Figure 5

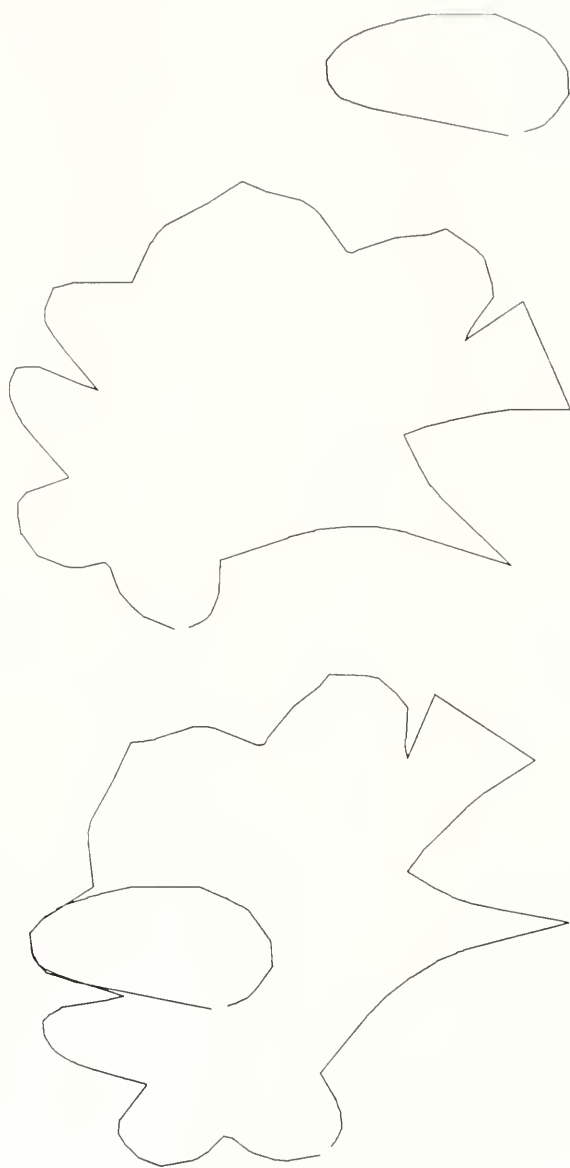


Figure 6



Figure 7



Figure 8

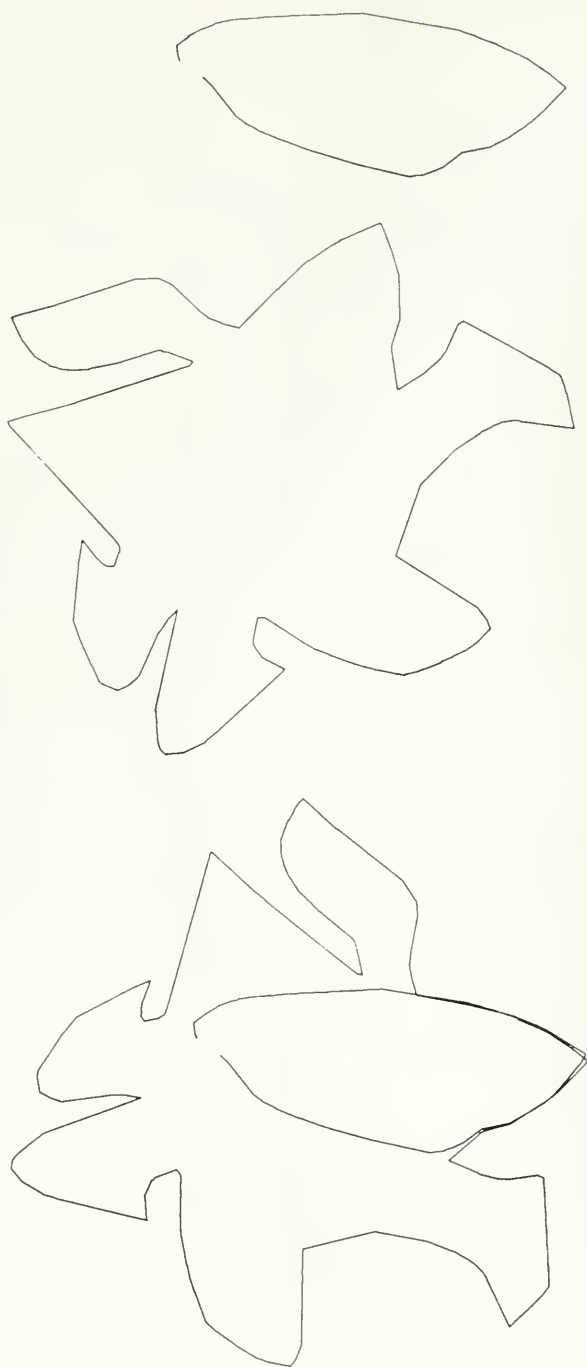


Figure 11

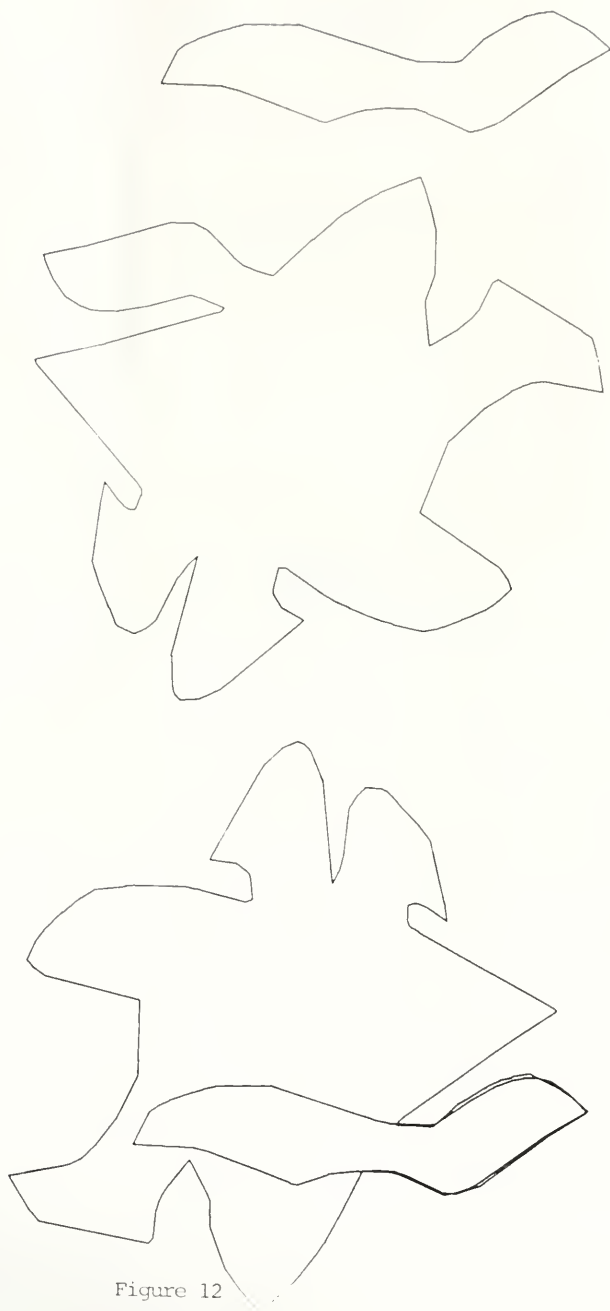


Figure 12

c.2

c. 2-2

MAN 2.

DATE DUE

GAYLORD			PRINTED IN U.S.A.

